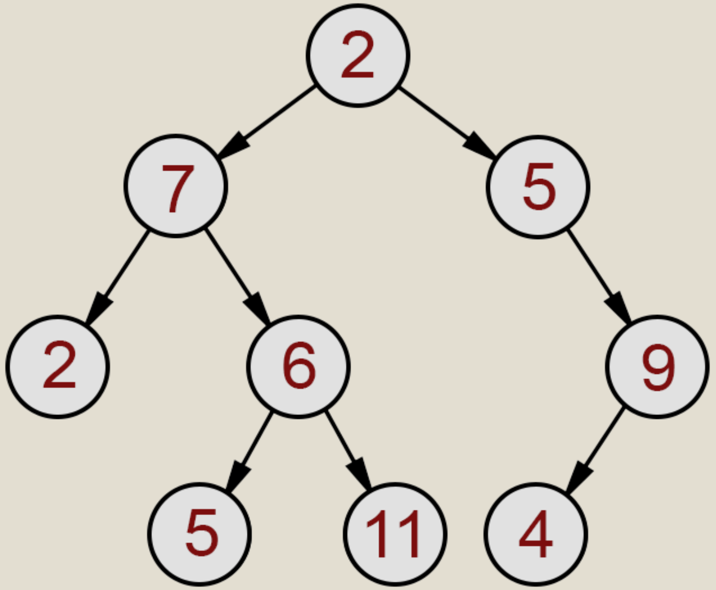
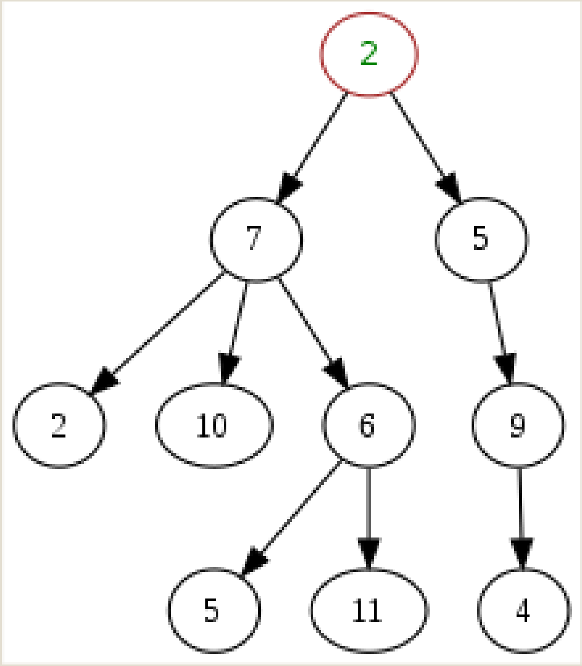
TREE

Tree

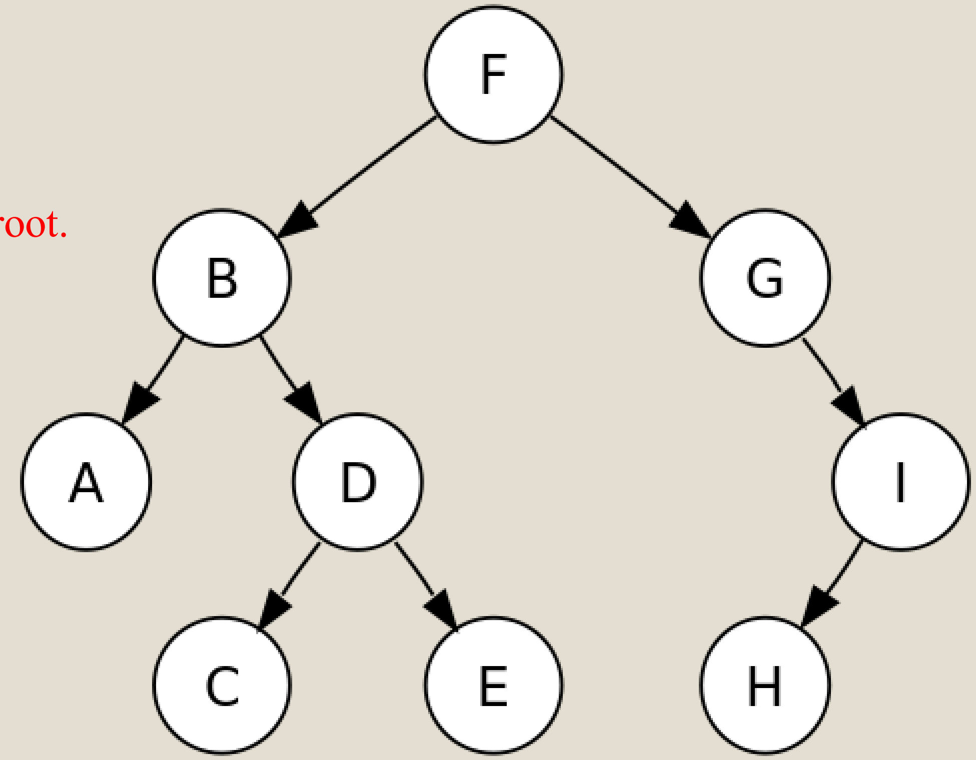
- It's a non-linear as well as non-primitive data structure.
- It's a collection of nodes and edges containing no loop and no cycle.
- Each node contains an item along with links to their immediate children.
- It also represents a hierarchical relationships between various elements.
- Applications of tree:
    - File system in computer
    - Better search operation
- Different variants of tree:
    - Binary Tree
    - B+ tree
    - Heap
    - Forest
    - Parse Tree
    - AVL tree
    - Red-black tree
    - B-Tree





Tree Terminologies

- **Node** – Each element of a tree is called as node.
- **Root -** The node at the top of the tree is called root. A tree has always only one root.
- **Edge –** Connection between any two nodes.
- **Parent-** Immediate predecessor of a node.
- **Child-** Immediate successor of a node.
- **Sibling-** Group of nodes having a common parent.
- **Leaf / Terminal node-** Node with no child.
- **Non-terminal node-** Node with single child or multiple children.
- **Path -** Path refers to the sequence of nodes along the edges of a tree.
- **Ancestor-** Node reachable by repeated proceeding from child to parent.
- **Descendant -** Node reachable by repeated proceeding from parent to child.



Nodes – A, B, C, D, E, F, G, H, I

Root – F

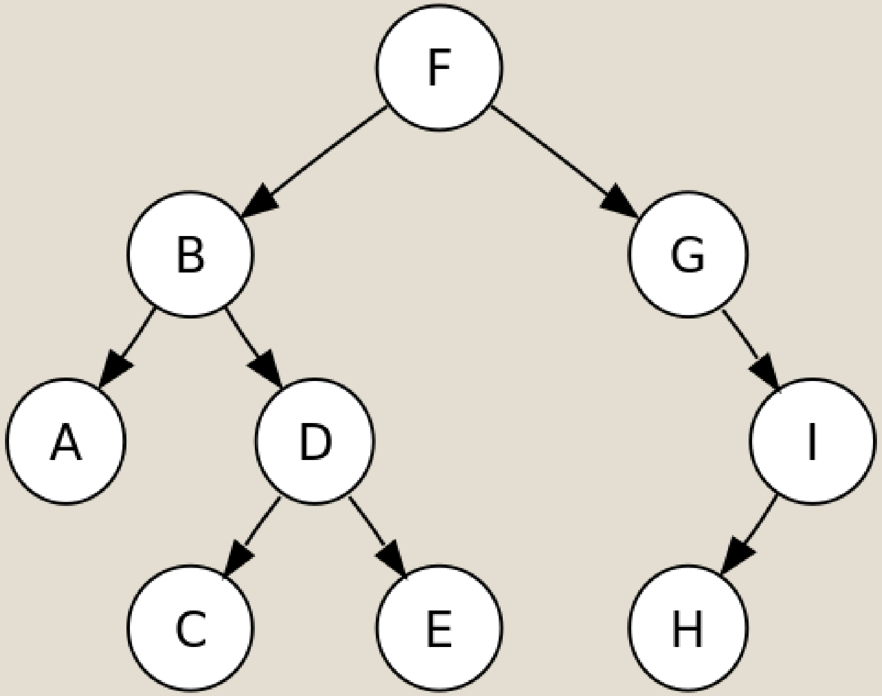
Leaves – A, C, E, H

Sibling – BG, AD, CE

Non-terminal nodes – B, D, G , I, F

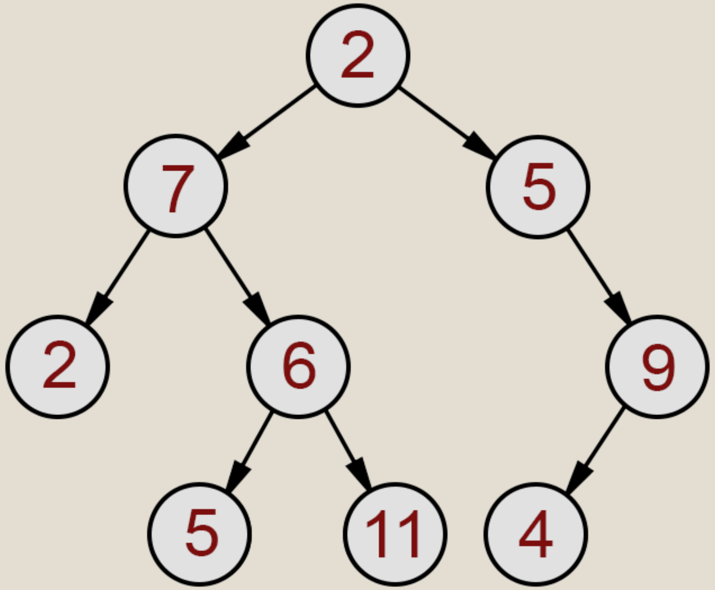
Tree Terminologies

- **Degree of a node-** Number of edges coming out of a node. Degree of leaf is always 0.
- **Degree of tree-** Maximum degree of a node is the degree of a tree.
- **Level-** Represents the generation of a node. Root is always at Level 0.
- **Height-** The number of edges on the longest path between a node and a descendant leaf. Leaf is always at Height 0.
- **Depth -** The distance between a node and the root. Root is always at Depth 0.



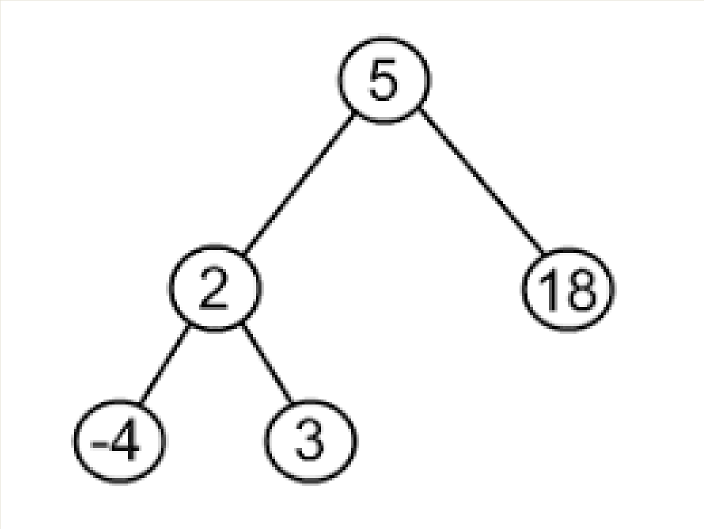
Degree of tree = 2

Node	Degree	Level	Depth	Height
F	2	0	0	3
B	2	1	1	2
G	1	1	1	2
A	0	2	2	0
D	2	2	2	1
I	1	2	2	1
C	0	3	3	0
E	0	3	3	0
H	0	3	3	0



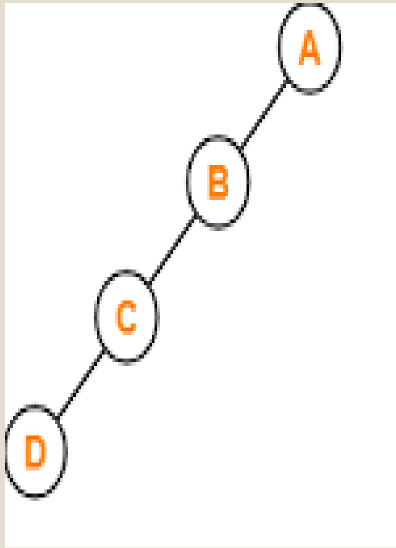
Degree of tree = 2

Node	Degree	Level	Depth	Height
2	2	0	0	3
7	2	1	1	2
5	1	1	1	2
2	0	2	2	0
6	2	2	2	1
9	1	2	2	1
5	0	3	3	0
11	0	3	3	0
4	0	3	3	0



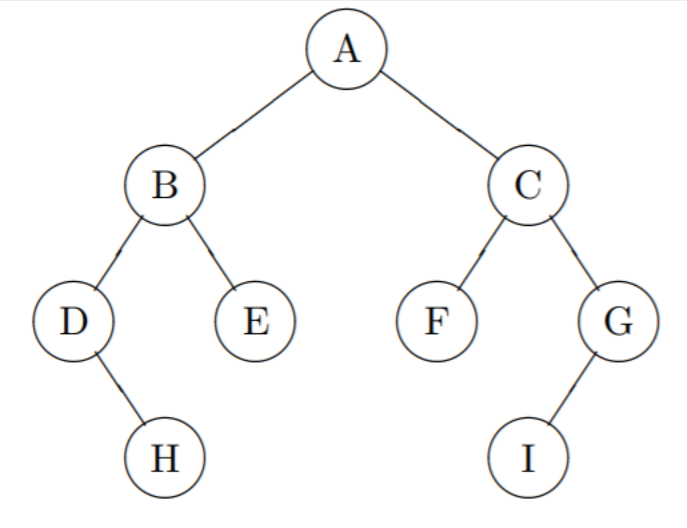
Degree of tree = 2

Node	Degree	Level	Depth	Height
5	2	0	0	2
2	2	1	1	1
- 4	0	2	2	0
3	0	2	2	0
18	0	1	1	0



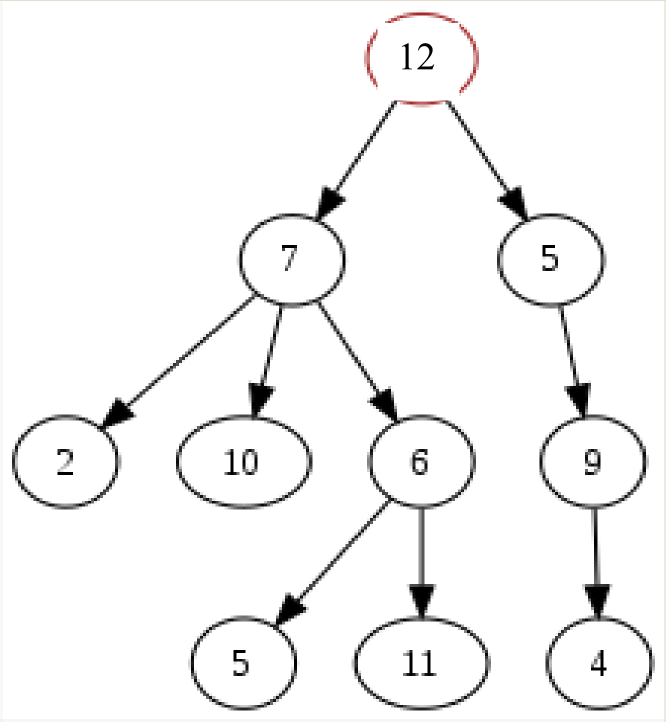
Node	Degree	Level	Depth	Height
A	1	0	0	3
B	1	1	1	2
C	1	2	2	1
D	0	3	3	0

Degree of tree = 1



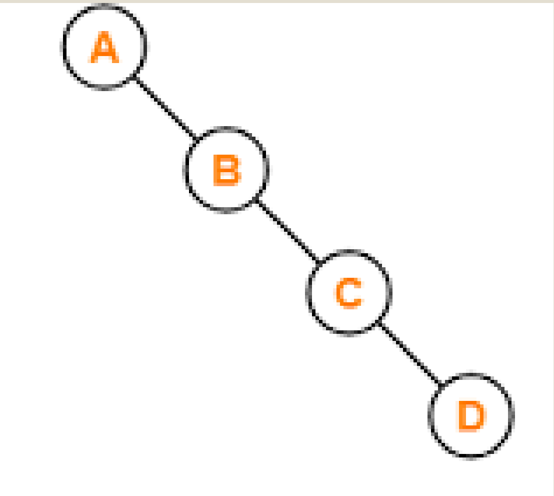
Degree of tree = 2

Node	Degree	Level	Depth	Height
A	2	0	0	3
B	2	1	1	2
C	2	1	1	2
D	1	2	2	1
E	0	2	2	0
F	0	2	2	0
G	1	2	2	1
H	0	3	3	0
I	0	3	3	0



Degree of tree = 3

Node	Degree	Level	Depth	Height
12	2	0	0	3
7	3	1	1	2
5	1	1	1	2
2	0	2	2	0
10	0	2	2	0
6	2	2	2	1
9	1	2	2	1
5	1	3	3	0
11	0	3	3	0
4	0	3	3	0

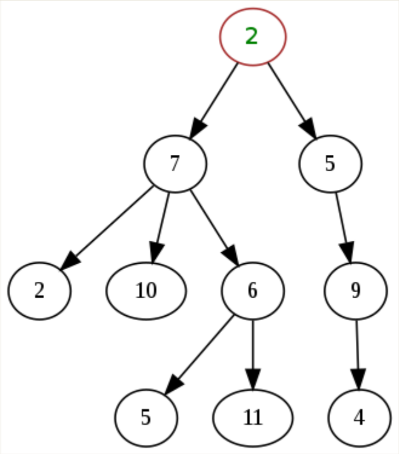


Degree of tree = 1

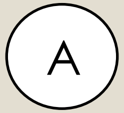
Node	Degree	Level	Depth	Height
A	1	0	0	3
B	1	1	1	2
C	1	2	2	1
D	0	3	3	0

Tree

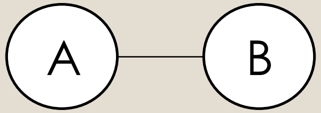
- Collection of nodes & edges.
- Contains no loop or no cycle.
- Doesn't contain any non-connected nodes.
- There is only a single root.
- A maximum of one edge connects two nodes.



Tree with multiple nodes



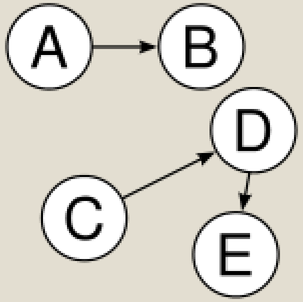
Tree with single node



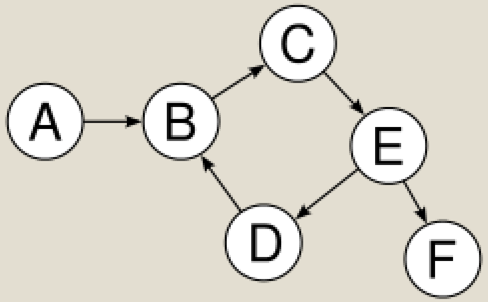
Tree with two nodes

Graph

- Collection of nodes and edges.
- Contains loops & cycles.
- Contains non-connected nodes.
- There is no root.
- Any two nodes can be connected by any number of edges.



Graph having non-connected nodes & two roots A & D



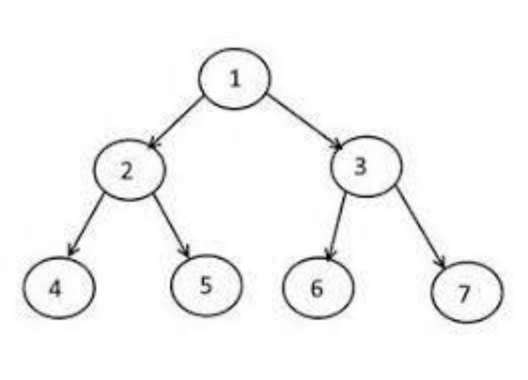
Graph having a cycle

Binary Tree

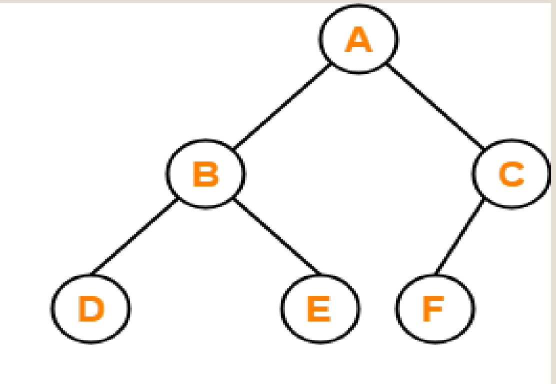
- Each node has at-most 2 children (May 0, 1 or 2 child/children).
- Maximum number of nodes at level l of a binary tree is $2^l - 1$.

Types of binary tree:

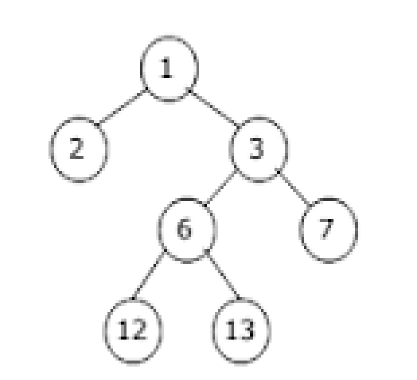
- **Full binary tree** – All leaves must be at same depth. A full binary tree of height h will contain a maximum of $2^{h+1} - 1$ nodes & 2^h leaves.
- **Almost complete binary tree** – All leaves are filled up from Left-to-right till the level $l - 1$.
- **Strictly binary tree** – Every node must contain either 0 or 2 children. Nodes can't contain 1 child in strictly binary tree. A strictly binary tree with n leaves always contains $2n - 1$ nodes.
- **Binary Search Tree (BST)** – Left node is smaller than the root and right node is greater than the root.



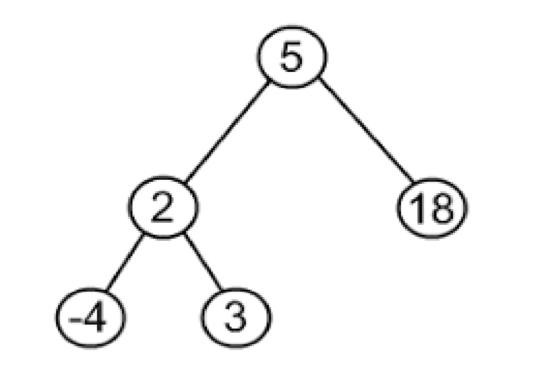
Full Binary Tree



Almost Complete Binary Tree



Strictly Binary Tree



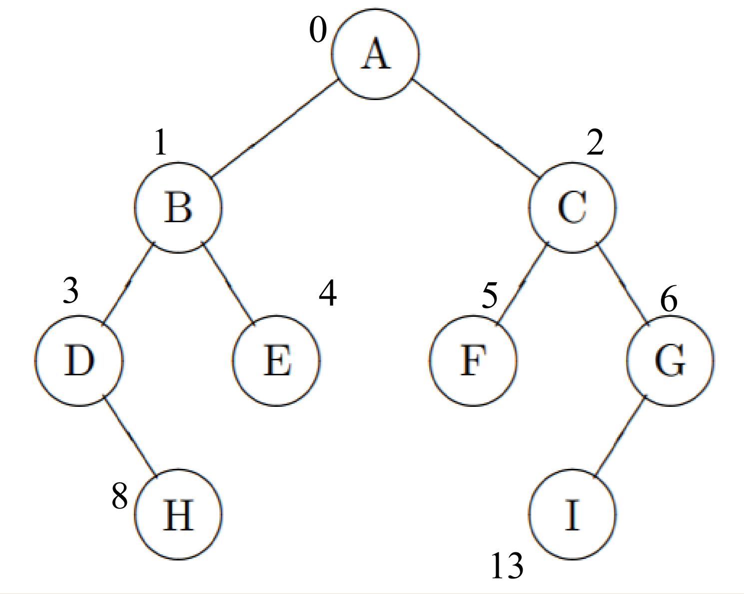
Binary Search Tree

Binary Tree Representation

- It can be represented using Array and Linked List.

Array Representation

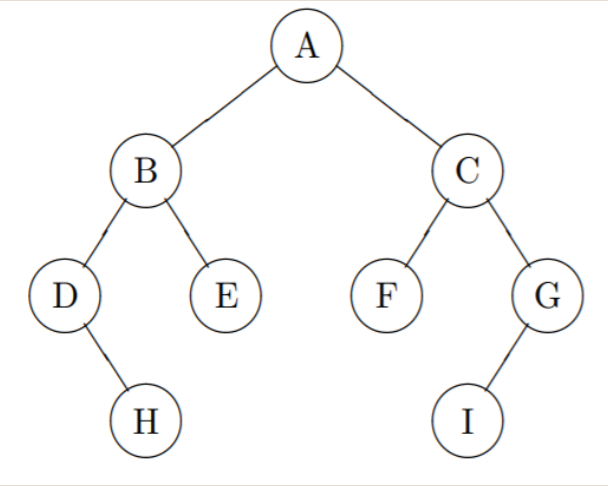
- Let, the array name is $tree$.
- Root is always at index $i = 0$, i.e., $tree\ [0]$.
- Left child is at index $(2i + 1)$, i.e., $tree\ [2i + 1]$.
- Right child is at index $(2i + 2)$, i.e., $tree\ [2i + 2]$.

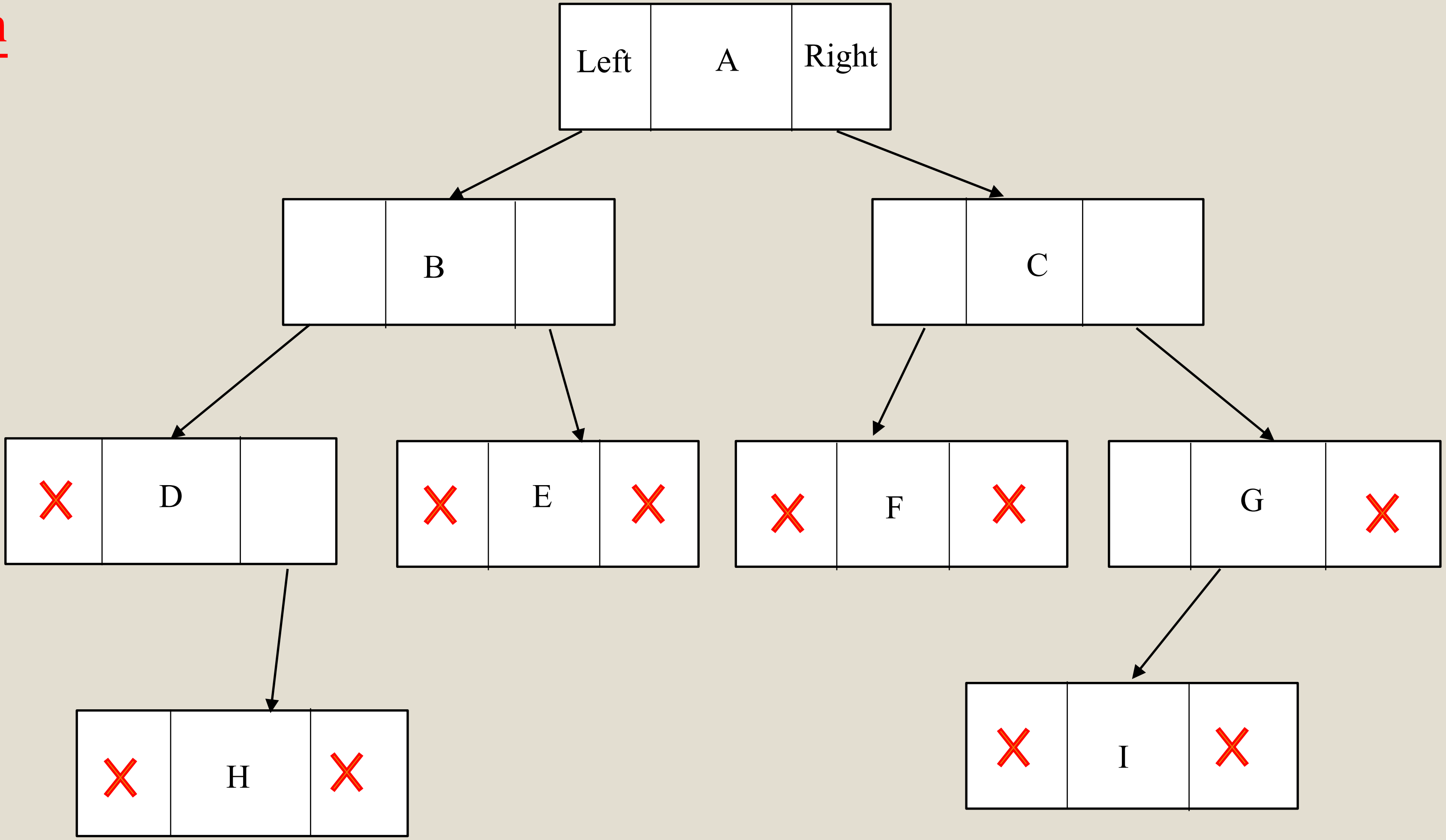


A	B	C	D	E	F	G		H					I	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Tree Representation

<u>Linked List Representation</u>





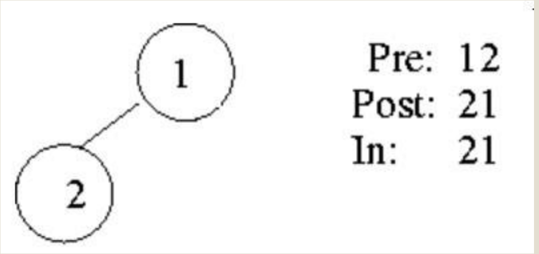
Binary Tree Traversal

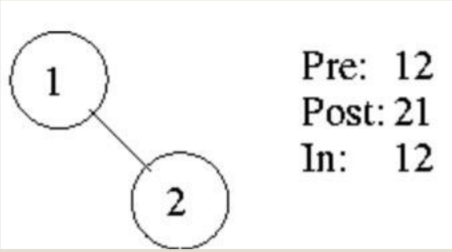
3 types of traversal: In-order, pre-order, post-order

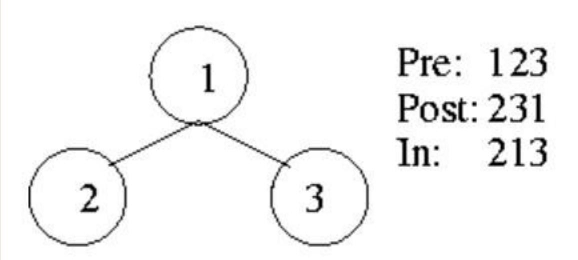
In-order **Rule: left subtree -> root -> right subtree**

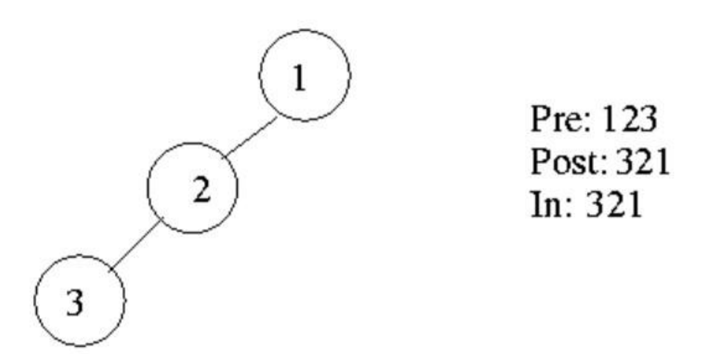
Pre-order **Rule: root -> left subtree -> right subtree**

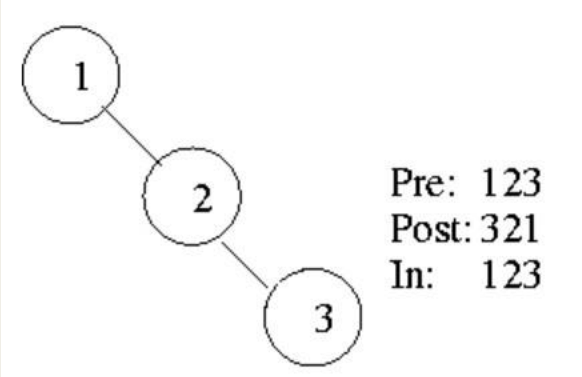
Post-order **Rule: left subtree -> right subtree -> root**

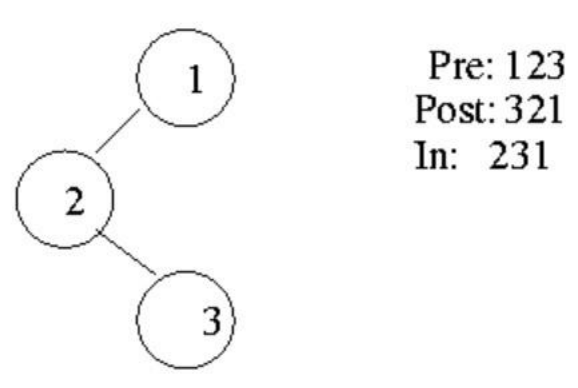


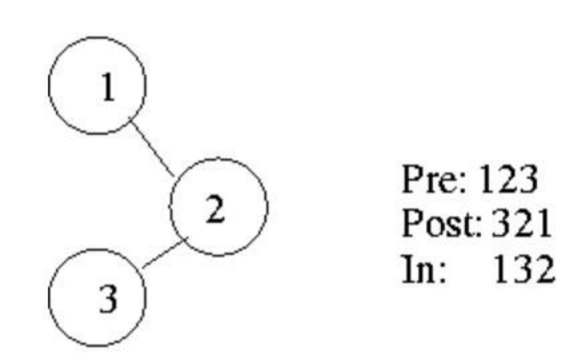


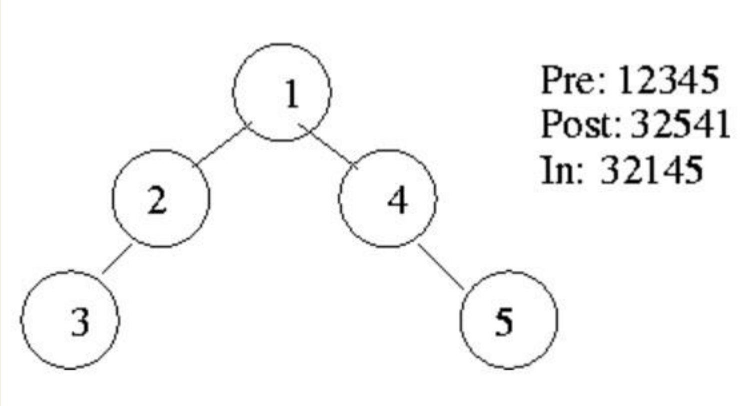




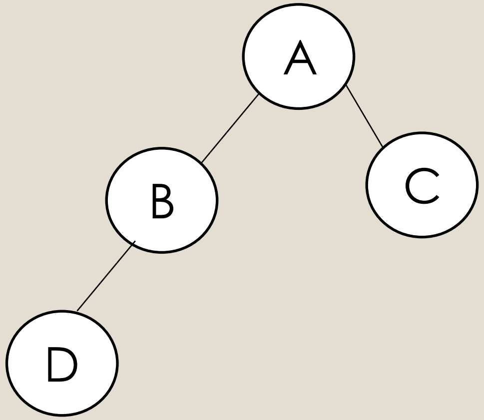








Binary Tree Traversal



In-order: DBAC

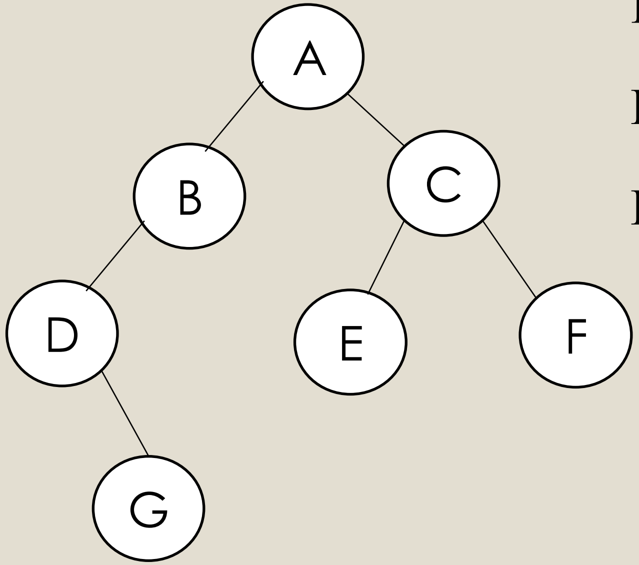
Pre-order: ABDC

Post-order: DBCA

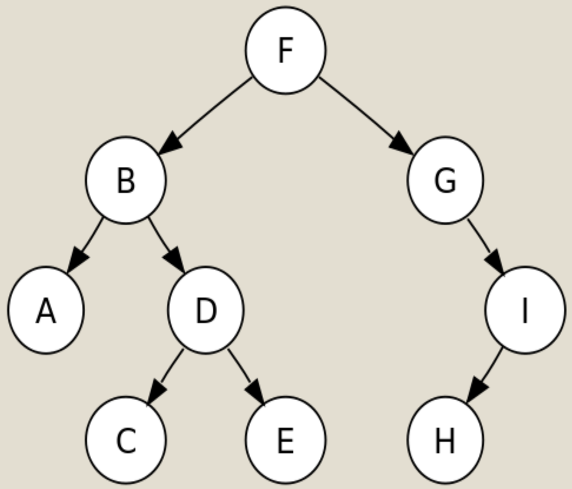
In-order: DGBAECF

Pre-order: ABDGCEF

Post-order: GDBEFCA



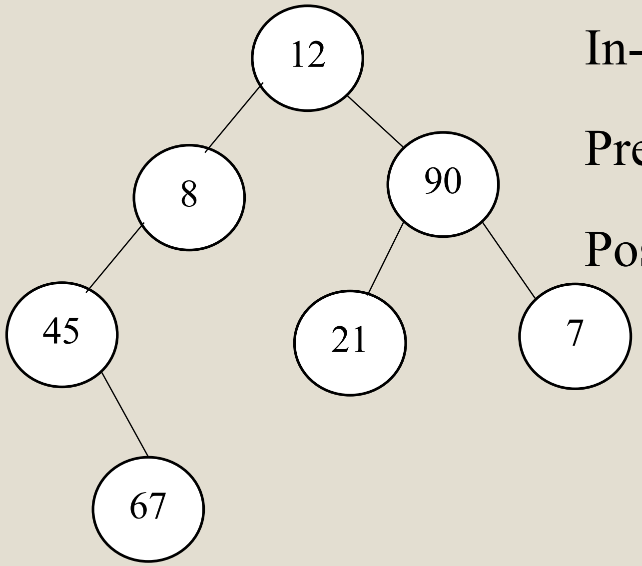
Binary Tree Traversal



In-order: ABCDEFGHI

Pre-order: FBADCEGHI

Post-order: ACEDBHIGF



In-order: 45,67,8,12,21,90,7

Pre-order: 12,8,45,67,90,21,7

Post-order: 67,45,8,21,7,90,12

Binary Tree Construction

In-order & Post-order Rules:

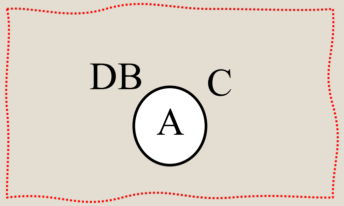
1. Scan the Post-order from Right to Left.
2. Last node of the post-order is the root.
3. Find the position of the root in the in-order list & find corresponding left & right subtree from the list.
4. Find the next parent from the post-order list.
5. Continue step 3 – 4 till all the nodes are processed.

Examples:

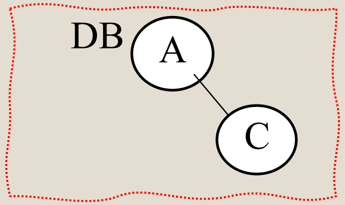
In-order: DBAC

Post-order: DBCA

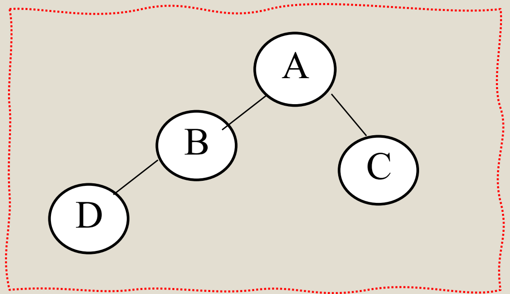
1. Since A is the last node in the post-order, A is the root. Left of A are DB & right of A is C the in-order list.



2. Next parent from post-order is C & it's right of A.



3. Next parent from post-order is B & it's right of D but left of A in the in-order list.
4. Next element from post-order is D & it's left of B.



Binary Tree Construction

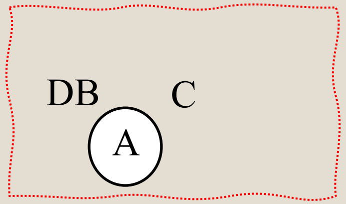
In-order & Pre-order

1. Scan the pre-order from Left to Right.
2. First node of the pre-order is the root.
3. Find the position of the root in the in-order list & find corresponding left & right subtree from the list.
4. Find the next parent from the pre-order list.
5. Continue step 3 – 4 till all the nodes are processed.

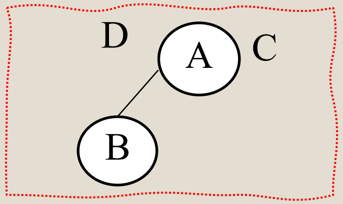
Examples:

In-order: DBAC

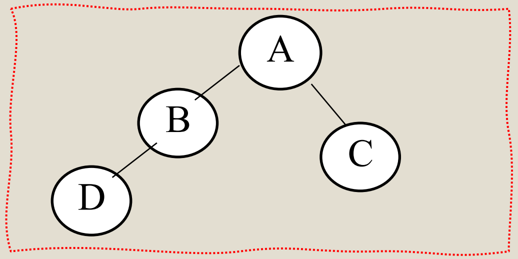
Pre-order: ABDC



1. Since A is the first node in the pre-order, A is the root. Left of A are DB & right of A is C the in-order list.



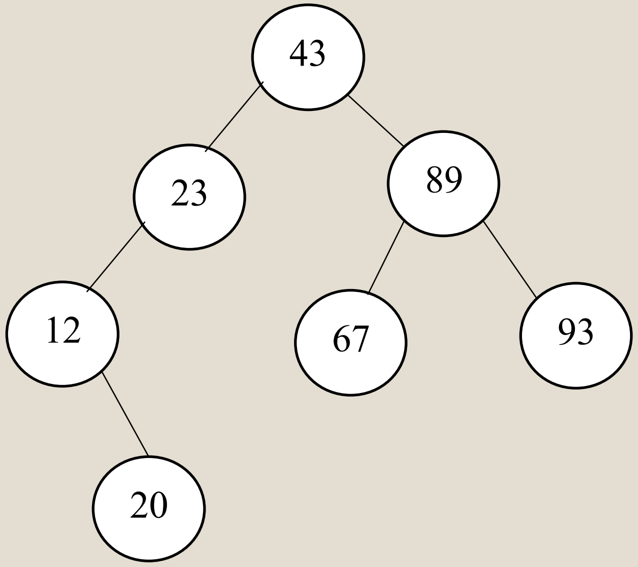
2. Next parent from pre-order is B & it's left of A.



3. Next parent from pre-order is D & it's left of A & B.
4. Next element from pre-order is C & it's right of A.

Binary Search Tree (BST)

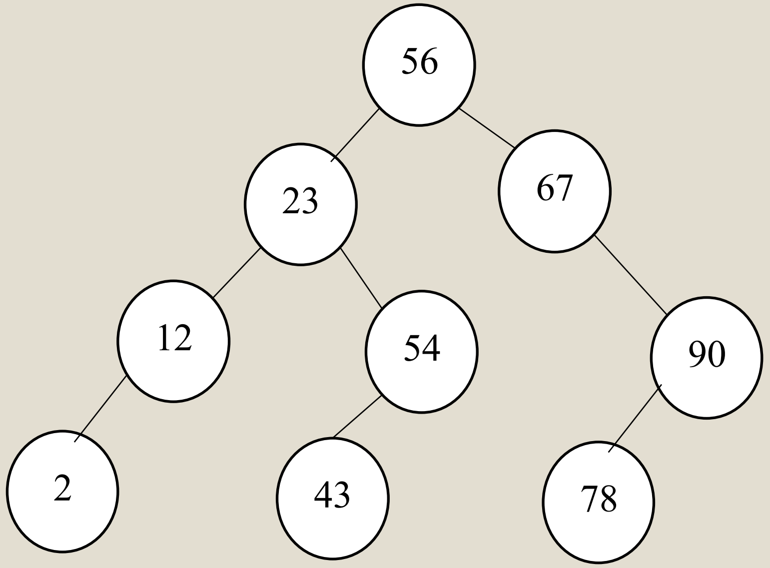
- Left child is always smaller than the parent.
- Right child is always greater than the parent.
- BST of n number of nodes:
  - Average case complexity of Search, Insert, and Delete Operations is $O(\log n)$.
  - Worst case complexity is $O(n)$



Construction of BST

1) Create a BST from the following list: 43,89,23,67,93,12,20.

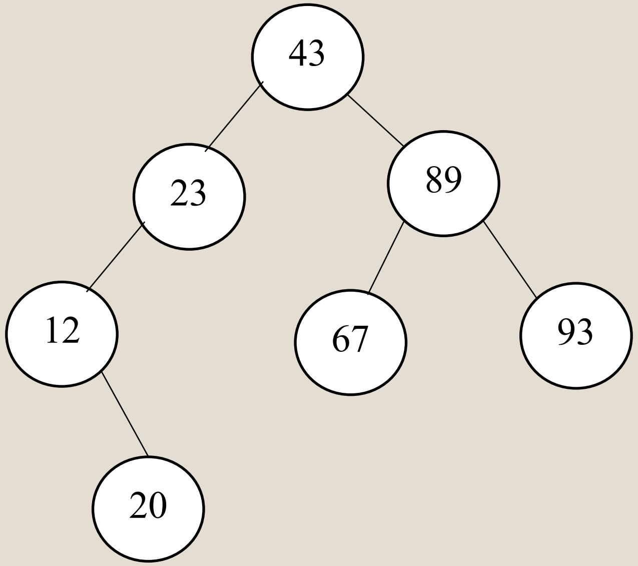
   1st node is the root. Find the left & right child as per the rule & Place them accordingly.



2. Create a BST from the following list: 56,23,67,54,90,12,78,2,43.

Binary Search Tree (BST) Traversal

Find In-order, Pre-order & Post-order traversal of the BST.

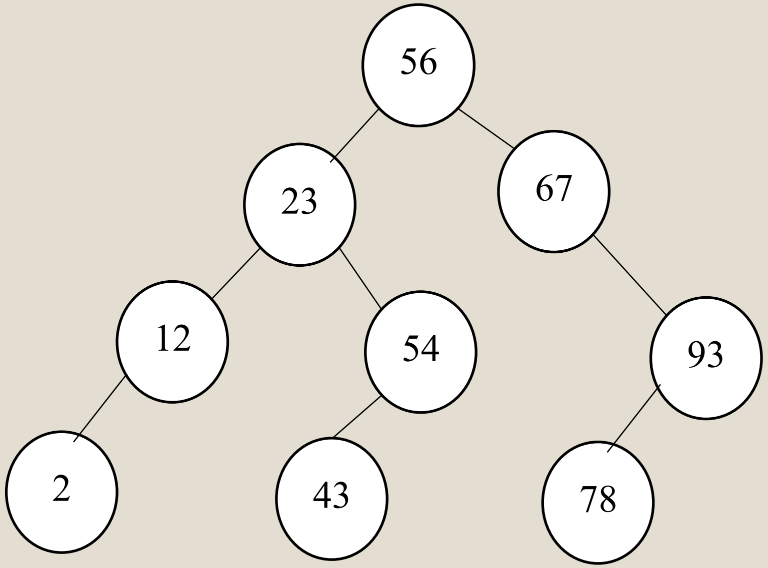


In-order: 12,20,23,43,67,89,93

Pre-order: 43,23,12,20,89,67,93

Post-order: 20,12,23,67,89,93,43

***In-order traversal of BST gives ascending order**



In-order: 2,12,23,43,54,56,67,78,93

Pre-order: 56,23,12,2,54,43,67,93,78

Post-order: 2,12,43,54,23,78,93,67,56